Copyright

by

Rodolfo Rosas

2013

The Report Committee for Rodolfo Rosas
Certifies that this is the approved version of the following report:

Cerberus: A Cloud-based Environmental Data Aggregator

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Adnan Aziz

William Bard

Cerberus: A Cloud-based Environmental Data Aggregator

by

Rodolfo Rosas, B.S.; B.S.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2013

Acknowledgements

I would like to thank Professor Adnan Aziz not only for supervising this report and guiding me through the whole process but for causing this idea to exist in the first place and providing me with the tools to do it. I would also like to thank Professor William Bard for his valuable time as reader for this report and for three of the most interesting classes I have had. And my thanks also go out to all the UT faculty and staff who have made this experience possible.

I would like to thank Cristina Fuentes for her invaluable help both in the initial stages of design and in the development of the web front end.

I would also like to thank everyone at IT Watchdogs for their help and cooperation in using their products as guinea pigs for this system.

Finally, no man is an island, and all that I do would not be possible without the continuous support of my family and friends. Thank you.

Abstract

Cerberus: A Cloud-based Environmental Data Aggregator

Rodolfo Rosas, M.S.E.

The University of Texas at Austin, 2013

Supervisor: Adnan Aziz

IT professionals must constantly ensure the integrity and availability of their services. One aspect to monitor is the physical environment surrounding their equipment. Factors such as high temperature, humidity or water leaks can all contribute to costly downtime. There are a variety of solutions for monitoring these factors but those which can make their data universally available are of particular interest. The more cost-effective alternatives on the market are found as embedded devices with integrated web servers. The drawbacks of these systems have been their resource limitations as well as the need for complex configuration to make their data available from any location. A solution based in the cloud could remove the resource constraints of embedded systems and easily achieve widespread availability.

This report presents Cerberus, a cloud-based environmental data aggregator that allows web server based devices to bypass their limitations and the necessary configuration to achieve widespread availability. It was implemented using two distinct cloud platforms and has been in active use for several months. Cerberus allows for environment monitors to push their data in as little as 250 ms while allowing for dynamic scaling as needed. The application provides all the necessary functionality such as alarm

generation, data visualization and archiving. The following presents the conception, design, implementation and future extensions to Cerberus as well as a comparative study of different cloud-based hosting services.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Chief among the concerns of IT professionals and data center managers is the need to reduce the downtime of their services. In today's world of digital economy, the inability to access a resource on the Internet could easily translate into heavy economic losses. These concerns gave rise to a market dedicated to prevent these potentially costly events. Among the solutions to this problem is the need to monitor the physical environment of hardware assets.

Environment monitors were created to measure various physical characteristics of sensitive equipment and its surroundings. By assessing the current temperature, humidity or quality of power as well as the presence of water leaks or other harmful conditions, these monitors can alert the necessary personnel as soon as a problem is detected. With this information, preventive actions can be taken before conditions deteriorate to the point of hardware failure. Furthermore, such monitoring capabilities could also be extended to any other industry where the environment is a factor, such as food and medication storage or even residential use.

Due to the nature of the problem, it is important that the information collected by environment monitors be always available and easily retrievable. To this purpose, many such products are designed as embedded devices with built-in web servers to supply the data. While being a practical solution, this raises a couple issues. First, embedded systems tend to be limited in resources. This means that a single device may not be capable of monitoring an entire installation and that multiple may be required, complicating the data retrieval process.

A second issue is the need for universal availability. An embedded web server may work fine inside a private network, but making the same server available from any

location requires either a costly static Internet address or advanced configuration. This report presents the conception, design, implementation and analysis of Cerberus, a cloud-based platform that attempts to address these issues while providing a seamless user experience. The process of creating Cerberus from design to fully functional implementation will be explored in the following sections.

1.1 CERBERUS

Cerberus aims to leverage existing cloud platforms to address the limitations of environment monitors. In order to better demonstrate how these limitations can be inconvenient, Figure 1 shows a typical system configuration. In this example, both local and remote computers need to access the current conditions in some installation. These conditions are being monitored by several environment monitors each on a different IP address within the installation's network. The local computer would have no issues connecting to each of the monitors, but it would be difficult to get the entire picture as each one must be accessed individually. At the same time a router acts as a gateway between the local network and the Internet. In order for the remote computer to access any of the monitors, incoming connections to the router would have to be forwarded to them. This would require the use of a different IP port number for each monitor and forwarding to be set up for each one. Additionally, the router’s external IP address would have to be either static or constantly updated using a service like Dynamic DNS so that the remote computer can reach it.

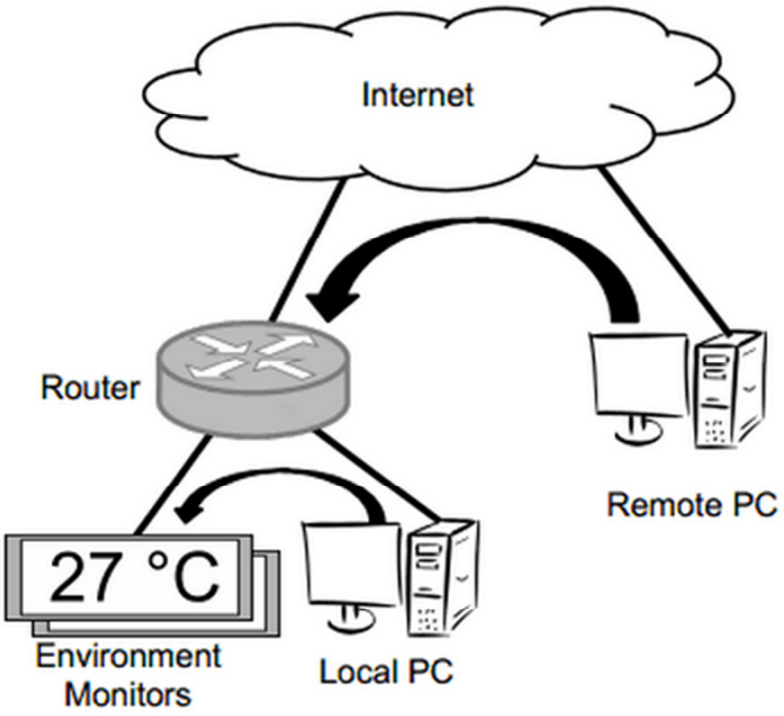


Figure 1: Typical network configuration

A cloud-based solution can eliminate the issues in a typical setting like the one in Figure 1. Since this kind of system can be made to have flexible resources, it would be able to receive, process, and store data from a large number of environment monitors. Having all the relevant data in a single location means that the user can be presented with a simpler interface where any issues can be immediately recognized and from where all other data can be easily navigated to. Furthermore, by having the environment monitors push their data to a single server, there is no need to access them remotely. This virtually eliminates any need for special network configurations as all that is needed is a connection to the Internet. Figure 2 shows an example of an installation using Cerberus.

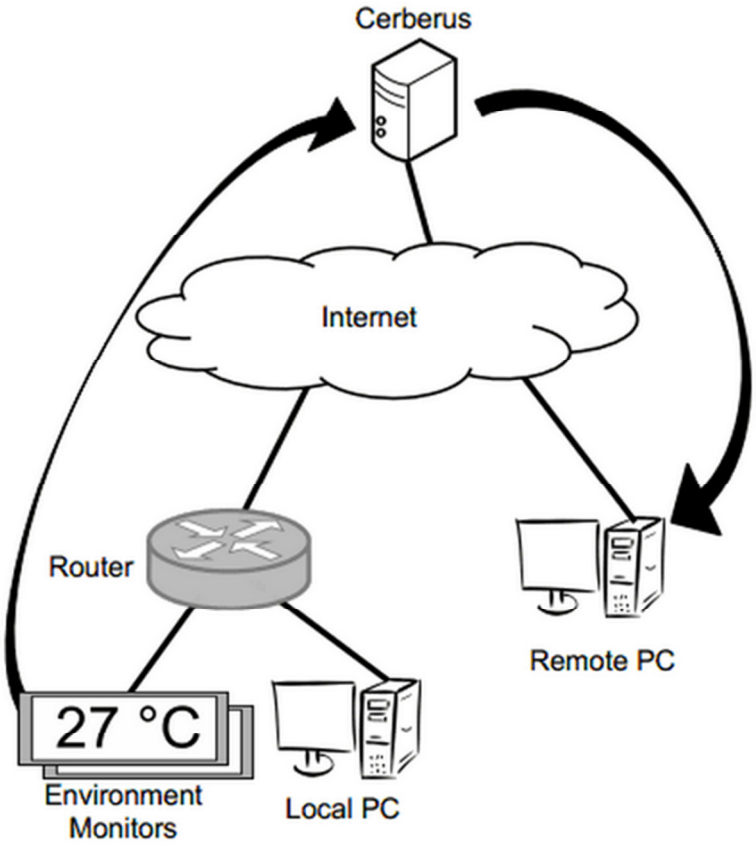


Figure 2: Cerberus configuration

An important difference between both scenarios presented above is the way in which data is obtained from environment monitors. Unlike a typical application where devices are polled directly, Cerberus will require them to push their data to a remote server. Furthermore, while the difficulties in network configuration for a typical system are in no way insurmountable, they are not trivial and could represent a problem to any non-technical users. By removing the need for this configuration and enabling the use of the cloud by default, Cerberus could make environment monitoring more attractive to these non-technical users or even users who do not have permission to access these configurations.

In addition to the benefits described above, a cloud-based monitoring solution could also expand on the feature set of embedded environment monitors. At a minimum, this platform would be capable of keeping historic values and plot them to identify trends or view a particular time range. It would also have alarm generation capabilities on the received data. This feature would compare the received values with user defined thresholds and perform some action if values are found to be unacceptable. While environment monitors already have this feature, Cerberus could expand on it by using online services like SMS text messages. As it stands, current solutions have to rely on Email to SMS services provided by cell phone companies to get these notifications. Having the option to send text messages directly through an internet service could help customers who do not have this feature yet desire a more immediate and noticeable notification.

1.2 USER STORIES

In order to better understand how Cerberus could be used, the following are examples of how different scenarios could incorporate the application. These user stories introduce the fictional character Bob and portray the usage of Cerberus in various settings. Bob works in the IT department of a large company and is responsible for making sure their servers are up and running. He uses a variety of environment monitors and remote sensors to keep track of the conditions inside their server rooms. If there are any issues, Bob can already receive email alerts and take action. Bob can also use the built in web interface on his devices to check their status at any time as long as he is on their same network. Bob is also a fine wine enthusiast and has an award winning collection in his summer home. He also manages real estate property on the side.

1.2.1 Information Technology

Bob would like to be aware of the conditions in his server room at any time during the day or night but his manager has placed strict policies regarding access to the internal network from the Internet. Bob learns about Cerberus and configures his environment monitors to push data to the service. Now he can log onto his account and see the latest measurements from all his sensors from anywhere in the world with no concerns about his company's network policies. Furthermore, he can configure alerts that send him SMS messages for when he does not have network access, a feature that his existing products do not offer.

1.2.2 Home users

Bob is worried about his wine collection at his summer home and wishes to protect his investment. He wants to constantly monitor the conditions of his wine cellar but his caretaker does not know anything about network configurations or maintenance. Instead of having to deal with the complex configuration, Bob sends an off-the-shelf environment monitor preconfigured to use Cerberus. The caretaker then just connects it to the network and places it in the wine cellar. With no additional effort, Bob can now access Cerberus and the conditions of his wine collection at any time.

1.2.3 Aggregation:

Bob must manage several apartment buildings. In the past, he has been able to detect critical problems in record time by using a web enabled environment monitor. This task is not easy since he has to access many individual web interfaces each representing

one of his installations and he is a busy man. He needs an easier way to view all the data in a single place and finds Cerberus. By configuring all his existing devices to send their data to Cerberus he can now view them all in a single central location and save some time. Furthermore, he can now configure alerts for each sensor from a single interface rather than doing it on each individual unit.

1.3 CONTRIBUTIONS

This report presents Cerberus as it is taken from conception to a finished application. Areas to be covered include:

- Vision and Design: Cerberus represents a solution to a real world need in the environmental monitoring market. This need can be met by developing a cloud based solution and it is explored and refined into a set of requirements, both functional and nonfunctional, and features that will make it desirable to the user.
- Implementation: The above vision was realized by employing the latest technology available on the market. Two fully functional versions of Cerberus were implemented using fundamentally different cloud platforms in an effort to select the best alternative. The application has proven to be stable enough to be in service for months while doing everything it was designed to do.
- Analysis: Qualitative and quantitative results were obtained by testing each implementation of Cerberus. Each alternative was analyzed in terms of its speed, costs, usability and other metrics relevant to its performance once fully deployed.

Cerberus is different from existing aggregation software in that it employs a cloud platform which allows it to easily scale and to be accessed from any location. It is also

different in that it allows for devices to push their data to a server rather than relying on them being constantly queried. Furthermore, no intermediate software is required for devices to make their data available. This allows for a system that will work requiring only Cerberus, an environment monitor and an internet connection between the two. No additional hardware or software is necessary.

1.4 OUTLINE

The remainder of this report will explore the design and development of Cerberus. Chapter 2, Cerberus Specifications and Requirements, will detail the specific objectives and behavior of the system. Chapter 3, System Design, presents a top level view of how the system is put together and what each component does. Chapter 4, Results, presents the outcome of the project and analyzes the performance of Cerberus. Finally, Chapter 5 presents the conclusions.

Chapter 2: Cerberus Specifications and Requirements

This chapter provides details on what Cerberus should do and how it works. It specifies the functional and nonfunctional requirements that define the interaction between the application and the environment monitors or users. Also, a mock user interface is provided as a guide for what will be implemented.

2.1 Functional Requirements

The following is a list of the basic functions that must be supported by Cerberus:

- Data Upload: Cerberus must be reachable by an environment monitor with Internet access. It must also be capable of establishing a connection with the monitor using TCP/IP and of receiving data from it using standard HTTP requests. Environment monitors must be capable of periodically sending this data in appropriate intervals. This means all important data events must be captured but excessive data volumes must be avoided.
- Data Processing: Cerberus must be capable of processing the received data to identify the environment monitor sending it, any attached sensors and all measurements associated with them, and any other data that may be deemed valuable. The data received should be encoded in a parsable file format such as XML or JSON.
- Data Persistence: Relevant information received by Cerberus must be stored in persistent storage for later use. Data to be stored should include the latest environment monitor configuration data and measurements as well as historic readings. This data should be easily and efficiently retrievable as well as persist until cleared by the application.

- Data Presentation: On user request, Cerberus must be capable of presenting all received data in the form of a web page graphical user interface. This interface should be easy to navigate and understand. Data shown will include both current information as well as historic values. The interface must be accessed through standard HTTP requests over TCP/IP.
- User Control: Users must log into Cerberus in order to view their data. Cerberus must control user access. User credentials must be properly handled and encrypted.
- Alarm Generation: The user must be able to configure alarms based on thresholds for measurements. Cerberus will compare received values to existing alarms and trip alarms when appropriate. These alarms will provide a set of actions to perform in the form of email messages or SMS text messages. When an alarm is tripped, messages will be sent by the system to the indicated recipients. When the alarm returns to the clear state, messages are also sent to indicate that the condition has been resolved.
- Environment Monitor Availability: If an environment monitor fails to push data to Cerberus for an extended period of time, that device must be marked as missing by the system. Any recipients configured for any alarms on a missing device will be notified of the situation so that communications can be restored.

2.2 GRAPHICAL USER INTERFACE MOCKS

Cerberus depends on a web front end graphical user interface to interact with the user. The following images are mock representations of what should be implemented in the final design.

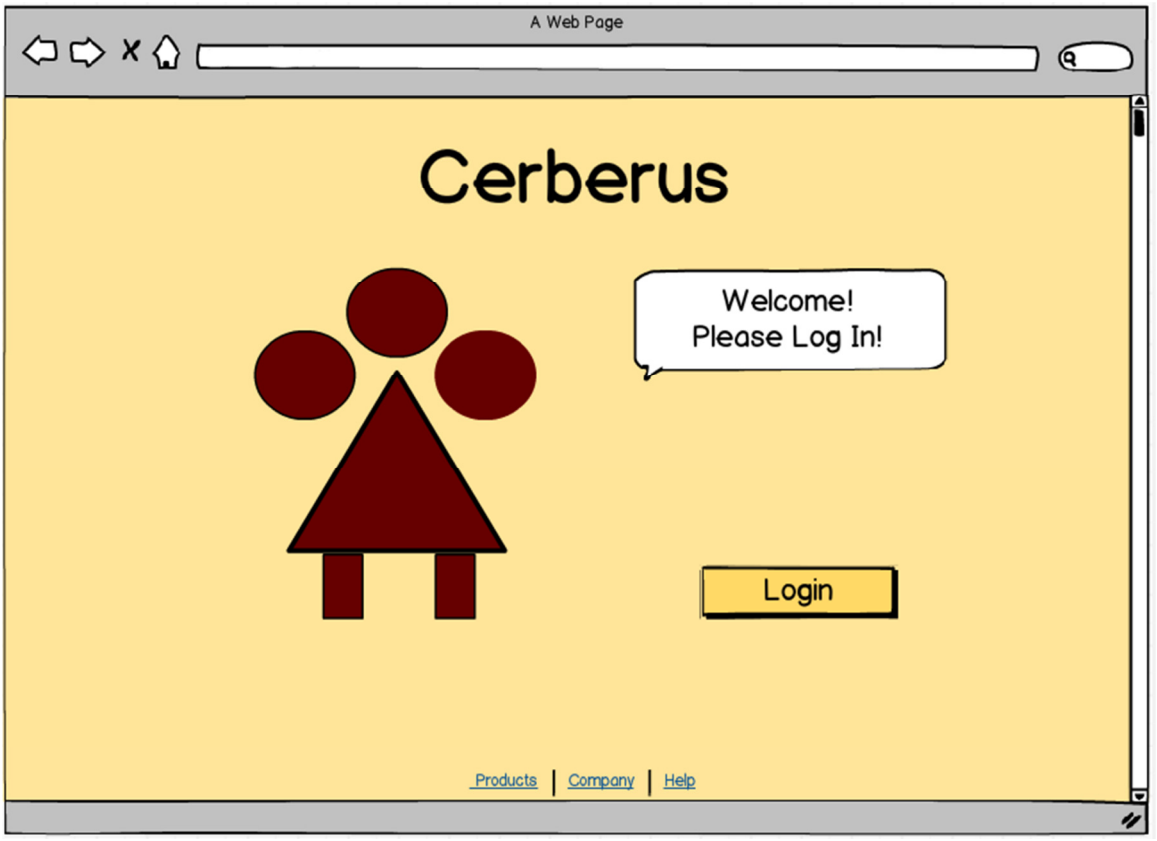


Figure 3: Initial welcome screen

Figure 3 shows the initial welcome screen to the application. This screen will show the Cerberus logo and will prompt the user to log in to the system.

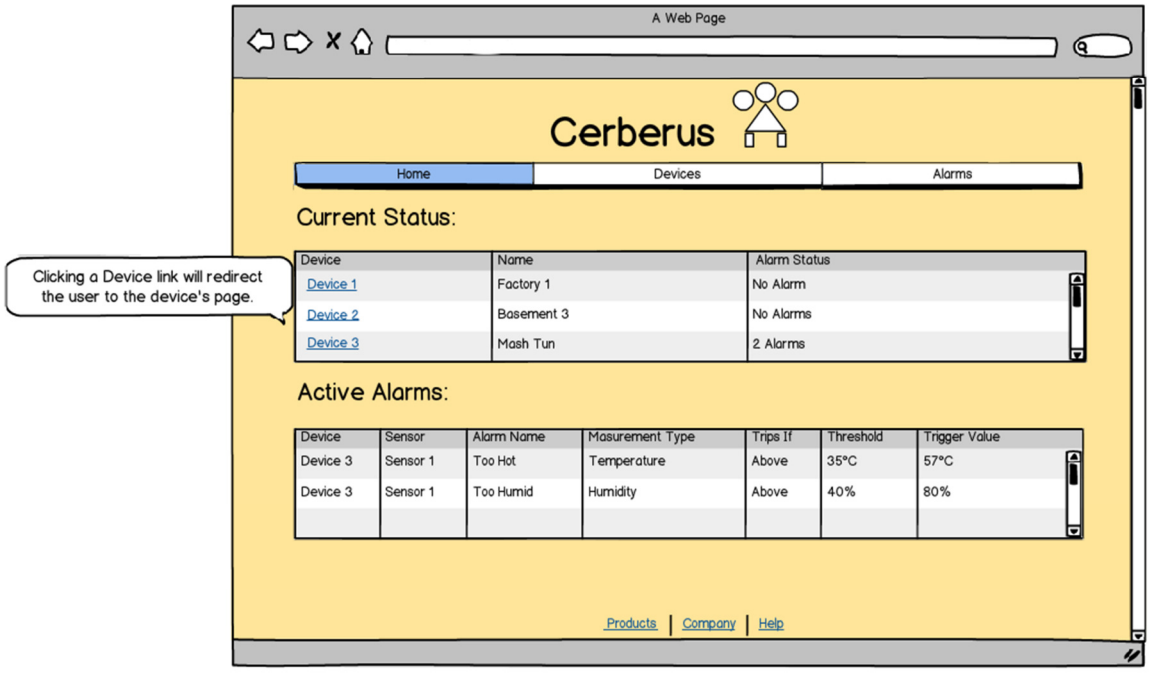


Figure 4: Main view

Figure 4 shows the main view screen for the applications. This page will have a list of all known environment monitors for the logged in client. The device list will show the name of each monitor and will indicate if there are any alarms currently tripped for it. Clicking on any item on the list would bring up the device view page for that item. In addition to the devices, this screen will also show a summary of all active alarms along with their details. This table will include the configured threshold as well as the last known value for the measurement. This page acts as an overall summary of available information and would allow users to see if there is a problem at a glance.

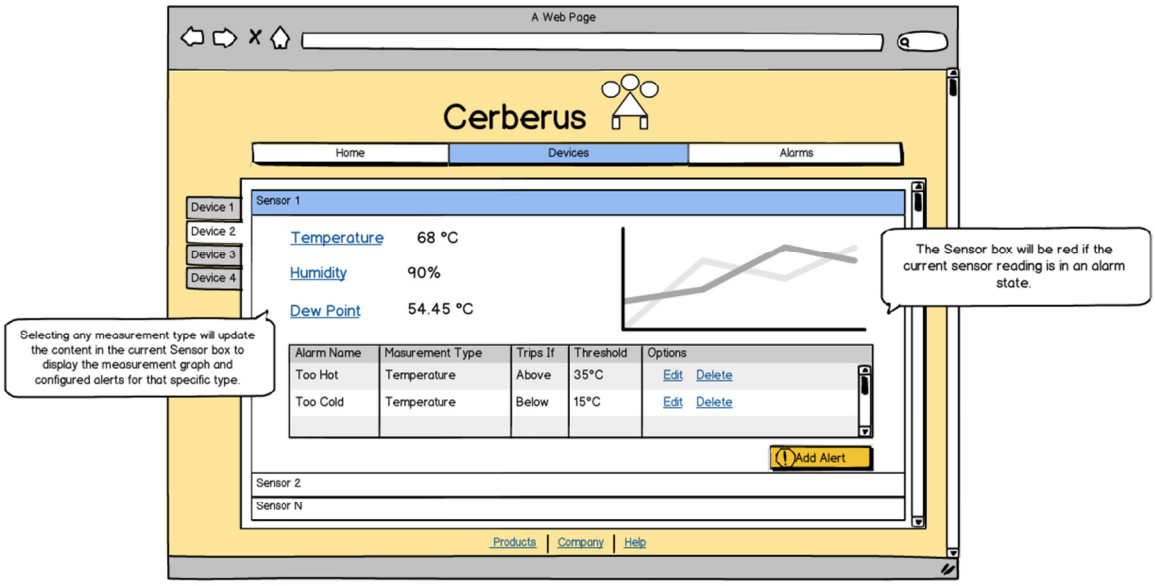


Figure 5: Device view

Figure 5 shows the device view page. This page shows the most recent readings for each environment monitor and each sensor plugged into them. The tabs on the left allow the user to navigate to each environment monitor while an accordion interface allows for individual sensor selection. All alarms relevant to the selected sensor are also displayed and those in the tripped state are highlighted in red. A plot showing historic data for the selected sensor's measurements is also displayed to the right. Clicking on an individual measurement shows its value and a plot with its historic data. Clicking on the "Add Alert" button or the "edit" link on an alarm will open the alarm creation window.

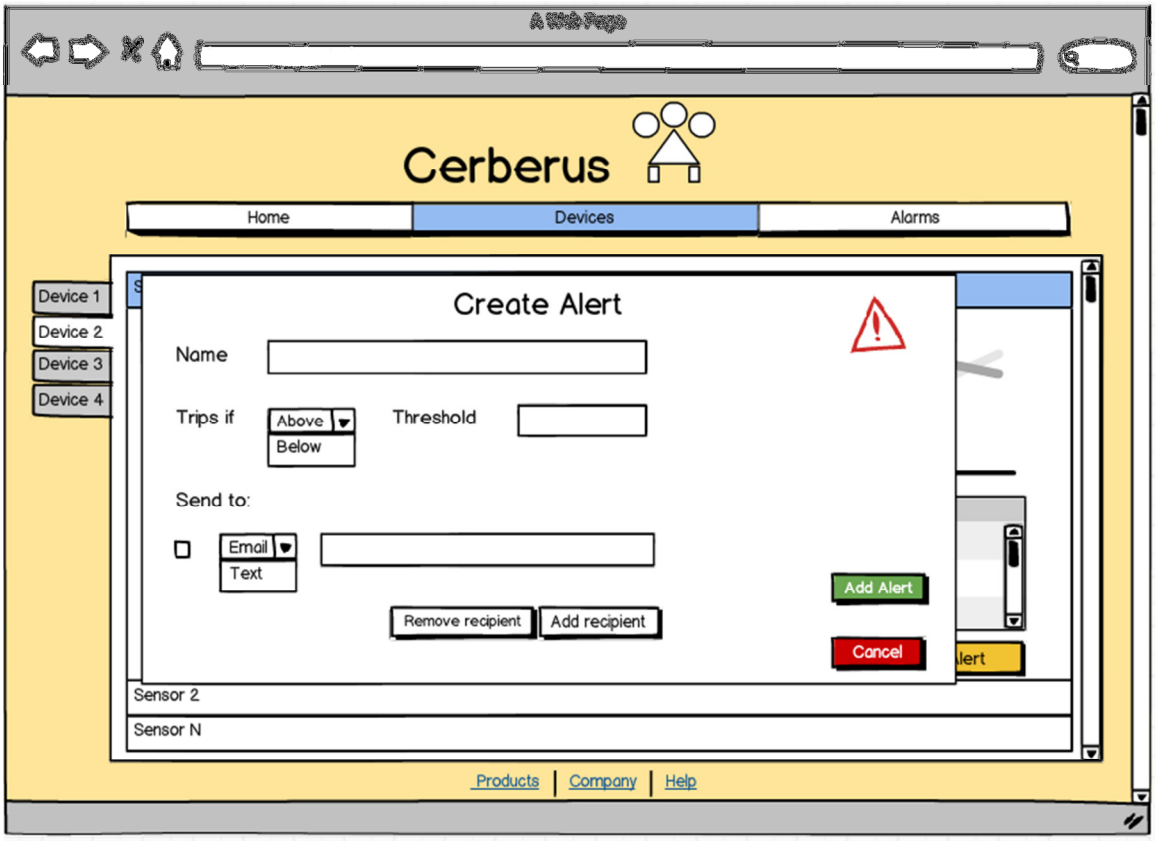
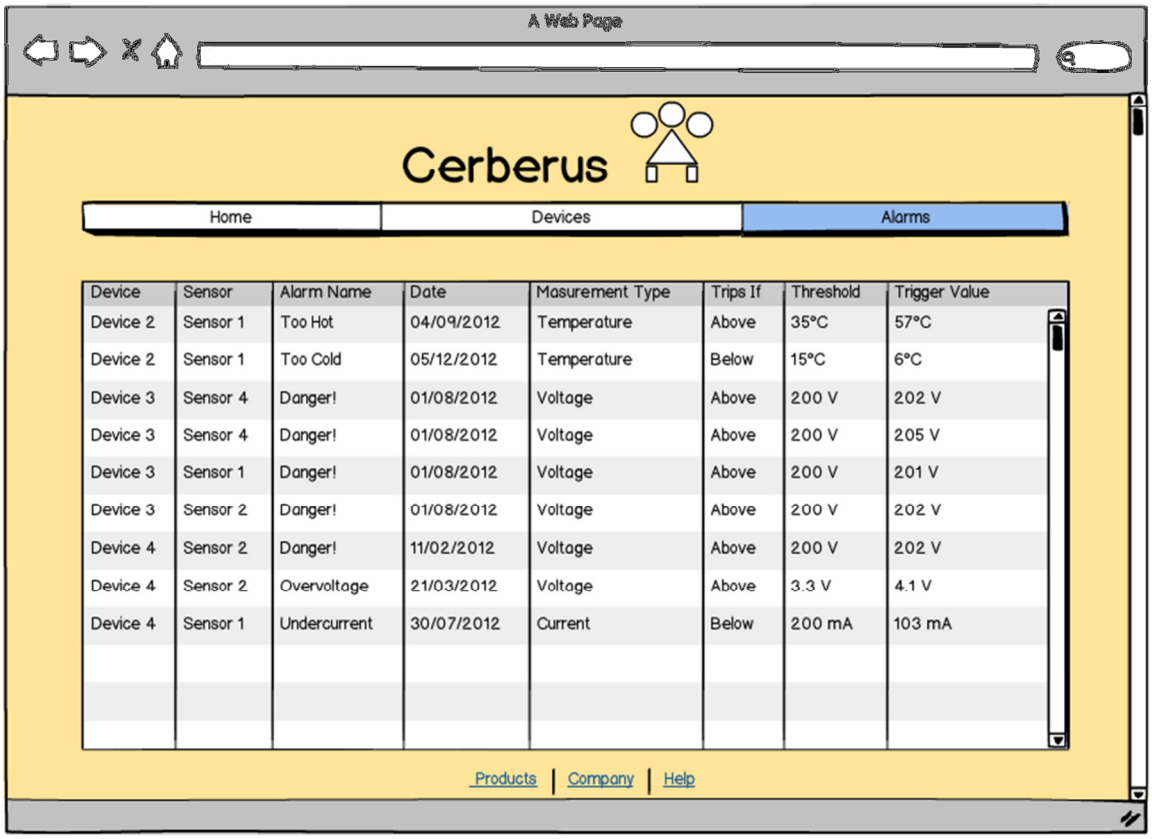


Figure 6: Alarm creation window

Figure 6 shows the alarm creation window. This window allows the user to create or modify an alert. Each alert can be configured with a name, threshold, trigger type, and list of actions and recipients in case the alarm trips. The actions and recipients can either be an email address or a phone number for SMS messages.



Device	Sensor	Alarm Name	Date	Masurement Type	Trips If	Threshold	Trigger Value
Device 2	Sensor 1	Too Hot	04/09/2012	Temperature	Above	35°C	57°C
Device 2	Sensor 1	Too Cold	05/12/2012	Temperature	Below	15°C	6°C
Device 3	Sensor 4	Danger!	01/08/2012	Voltage	Above	200 V	202 V
Device 3	Sensor 4	Danger!	01/08/2012	Voltage	Above	200 V	205 V
Device 3	Sensor 1	Danger!	01/08/2012	Voltage	Above	200 V	201 V
Device 3	Sensor 2	Danger!	01/08/2012	Voltage	Above	200 V	202 V
Device 4	Sensor 2	Danger!	11/02/2012	Voltage	Above	200 V	202 V
Device 4	Sensor 2	Overvoltage	21/03/2012	Voltage	Above	3.3 V	4.1 V
Device 4	Sensor 1	Undercurrent	30/07/2012	Current	Below	200 mA	103 mA

Figure 7: Alarm view

Figure 7 shows the alarm view for Cerberus. This page can be accessed using the top navigation bar and shows all the alarms in the system. Any alarm that is currently tripped will be shown in red.

2.3 Non-functional Requirements

The following is a list of additional requirements for Cerberus to function as intended:

- Availability: Since Cerberus will be receiving data from a multitude of environment monitors, it should always be available to process requests. Any

amount of time for which Cerberus is unavailable would represent missed data and the potential to miss an important event.

- Scalability: One of the basic points of moving an aggregating solution to the cloud is to allow for a dynamic demand on the system. As such, Cerberus should be able to easily scale with demand. This includes handling cases where a multitude of environment monitors happen to start sending their data at the same time.
- Response Time: Cerberus should be able to respond in a reasonable time when dealing with either environment monitors or end users. In the case of environment monitors, this means that connections should be promptly established and data transferred, processed and stored in a timely manner. From a user perspective, Cerberus must appear interactive and snappy.
- Cost: In order to properly market Cerberus, its recurring costs should be sustainable and appropriate for the amount of devices supported.
- Maintainability: Not only should Cerberus be well structured and easy to maintain as problems arise, but it should also be able to easily integrate additional internet services as needed. From an engineer's perspective, updating the user interface, replacing a deprecated service, or adding additional alarm functionality should not result in extended downtime or service degradation.

Chapter 3: System Design

This chapter provides details on how Cerberus was assembled, what are its components and how they interact.

Figure 8 shows a top level view of the resulting Cerberus system and each of its components. While some changes were required for the environment monitors, all other items outside the box labeled as Cerberus are provided by a third party. As such, the main focus of this report resides with those components that make up the Cerberus system. The following sections provide more information as to what the function of each component is and how it accomplishes its task.

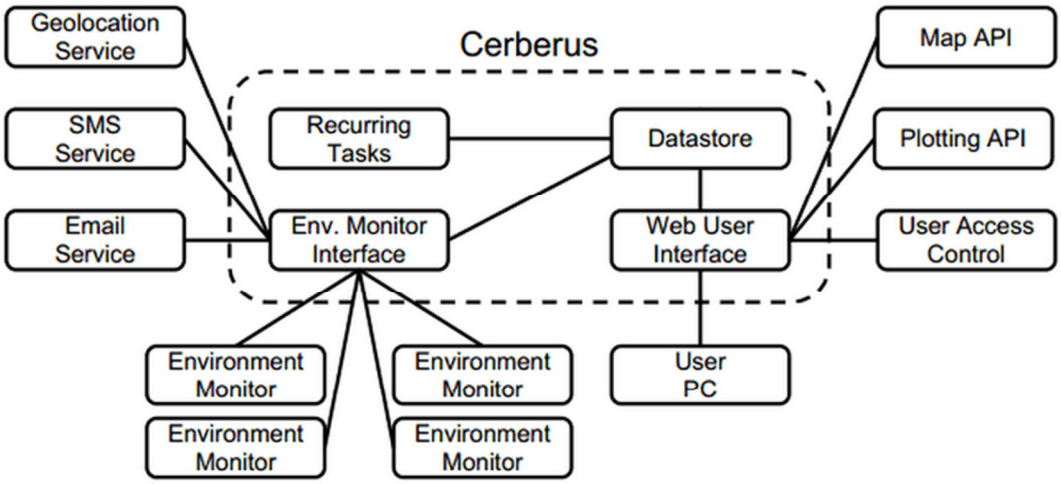


Figure 8: Cerberus components at a glance

3.1 Environment Monitors

The environment monitors represent the source of data for Cerberus. Due to the differences in design among environment monitors, a particular brand and model had to be chosen for initial implementation. The product selected was the WatchDog 15 [1]

designed and manufactured by IT Watchdogs [2]. This device provides built in temperature, humidity and dew point measurements as well as expansion ports for additional sensors.

As with most environment monitors, the WatchDog 15 is designed to respond to data requests initiated by a user. This product provides a URL from which an Extensible Markup Language (XML) formatted file with all the current measurements and alarm configurations may be downloaded. This design is not only incompatible with Cerberus, but it is the very limitation the system tries to address. As such, modifications were needed on the embedded server so that the XML file could be pushed to the cloud.

After testing various approaches, it was concluded that the best method would be to allow the WatchDog 15 to open an HTTP connection to Cerberus on a periodic basis. During each of these connections, the device uploads the XML file via an HTTP POST using a multipart/form-data content type. Furthermore, since the embedded web server contained within the product does not have enough buffer space to hold the entirety of the uploaded file, it needs to be generated on the fly. This means that the size of the posted file is not known in advance and chunked data encoding is required.

The period selection for the data upload was not trivial. Too short of a period has the potential of producing an overwhelming amount of data on the Cerberus server and of interfering with the regular tasks of the environment monitor. On the other hand, a long period has the potential of delaying or altogether missing the processing of data which would cause an alarm. A period of 10 minutes was finally selected as a suitable compromise. In order to further improve the responsiveness of the system and the timely response to alarm conditions, the data is uploaded immediately if an alarm changes state. Thus, data is sent every time an alarm trips or clears or if 10 minutes have elapsed since the last upload.

3.2 CERBERUS CLOUD PLATFORM

A cloud platform, in this scenario, is a service that provides access to cloud computing while simplifying its management and supplying additional features. In essence, it is a convenient way to abstract the infrastructure requirements of an application. This would be represented by the box labeled Cerberus in Figure 8. It is the platform that hosts all other components of Cerberus and provides all the necessary resources like a web server or data storage. While there are several alternatives when it comes to cloud platforms, Cerberus explores two widely used options, Google App Engine [3] and a simple Linux based server implementation with the Apache Tomcat webserver [4]. The intent is to compare a proprietary platform like Google's with more flexible alternatives represented by the Linux server. For the initial implementation of Cerberus, Amazon Web Services [5] was selected as the cloud platform to host the Linux application. Also, Java servlets were used with the Apache Tomcat webserver as a way to handle requests and generate dynamic content.

The rationale for using two distinct cloud platforms stems from the inherent differences and advantages with each. On one hand, Google App Engine allows for rapid development and deployment via custom plugins for popular IDE's as well as a wealth of built in services like their datastore and user service. Furthermore, this platform provides automatic scaling as the data or users increase. On the other hand, a Linux server allows for the most flexibility as the developer is not constrained to a single environment. While this does mean that scalability is no longer automatic, using a flexible platform like Amazon Web Services does make it relatively simple. This platform provides servers of increasing capabilities based on the needs of the system. At any point during operation,

extra servers can be quickly commissioned and services for dynamic load balancing or database servers are readily available. In short, the choice of cloud platform becomes a matter of convenience versus flexibility and the right answer can be determined by testing Cerberus on both.

3.3 CERBERUS ENVIRONMENT MONITOR INTERFACE

The environment monitor interface is the part of Cerberus that receives and processes data pushed from environment monitors and stores it in the datastore. It is implemented as a single Java servlet which responds to an HTTP POST on a particular URL. This interface expects an XML file with a particular format and parses it to obtain all relevant fields from the environment monitor. The tasks performed by this servlet can be divided into three general categories, environment monitor server maintenance, measurement collection, and alarm generation.

- Environment Monitor Server Maintenance: When parsing the XML file, the data interface first determines which environment monitor server is sending the data. The server is then either added to the system or updated and the list of known alarms for it is checked. New alarms are added and alarms that have been deleted on the server are removed from Cerberus. Additionally, if the server is new or its IP address has changed, Cerberus queries a geolocation service to determine the approximate physical location of the unit. The geolocation service used is Geobytes IP locator [6] and it is accessed using an HTTP GET query which returns the approximate longitude and latitude of the environment monitor's IP address. The location information returned is in JSON format and also includes

the geographic region and time zone which could be used for additional features later on.

- Measurement Collection: Once the server has been updated, the data interface servlet processes the latest measurement information. A separate device object is created or updated on the system for each sensor attached to the environment monitor or each one that has been removed since the last update. Each of these devices will, in turn, be referenced by a set of current measurement objects. The datastore entries for all devices and current measurements are updated to reflect the latest received XML file. Additionally, new entries are created for historic measurement values.
- Alarm Generation: Having stored all the measurement readings, the data servlet proceeds then to evaluate any alarm conditions. The set of latest values is compared to any alarms configured locally on Cerberus and the states are changed accordingly. Any actions configured for alarms which change state are also performed at this stage. SMS messages are sent using Twilio SMS service [7] by using their Representational State Transfer or RESTful API. This API is accessed by simply sending an HTTP POST to a URL tied to the SMS send action with fields specifying the content of the message and the recipient. Emails are sent by using Java APIs supplied by the system back end. The alarm status is also updated in the datastore.

3.4 CERBERUS DATASTORE

The datastore is the part of Cerberus that stores and serves up data upon request from the other components. The specific implementation of the datastore varies based on

the cloud platform used for Cerberus. In any case, this component is meant to provide scalability and the ability to easily access the required elements.

The Google App Engine version of Cerberus uses their built in datastore service [8]. This system is accessed through a special set of Java API calls that provide functions for reading and writing. This datastore provides atomic transactions, high availability and scalability by automatically replicating the data across multiple data centers. While this approach does not provide the same features as a relational database model, it does meet all of the needs for Cerberus.

Linux based alternatives to Cerberus employ a standard relational database such as MySQL. In these cases, the datastore can be accessed using standard database connector classes for Java and all essential database properties, like atomicity, are maintained. While having the database server live on the same machine as the rest of the application is not scalable, providers like Amazon Web Services do offer special database servers that can be used to serve a dynamic range of application servers.

Figure 9 shows a diagram of the schema followed by both versions of the datastore. The schema is implemented in the relational database model by using external references in the tables and unique fields where necessary. In the case of the Google App Engine datastore, the scheme is implemented by assigning a hierarchy to the keys for each stored object. This hierarchy is specified as a path indicating all the ancestors for a key. The hierarchy used in either model allows Cerberus to easily and efficiently retrieve, for example, all current measurements for a given device or all alarms related to a single server. It is worth noting that the schema provides a one-to-many relationship between servers and devices, devices and measurements, and each of those fields to an alarm.

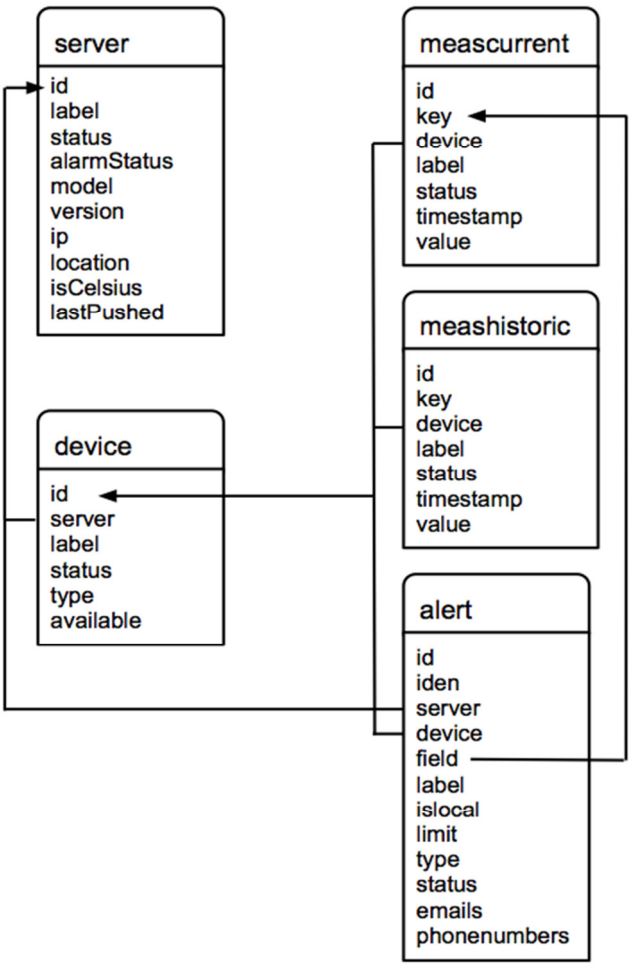


Figure 9: Datastore schema

3.5 CERBERUS RECURRING TASKS

Recurring tasks are necessary to monitor events that cannot be triggered by receiving data in the data interface. The most important example of one such event is determining when an environment monitor has gone offline. Since the lack of incoming data fails to trigger the data interface, a recurring task is used to check if too long has elapsed since the last data push by an environment monitor. In the case that the time

threshold has expired, the device is flagged as missing and messages are sent to all alarm recipients configured for the device to notify them of the situation.

Google App Engine provides support for recurring tasks by configuring a request that is to be executed every certain amount of time. In order to support this feature, this task was also implemented as a Java servlet that responds to an HTTP Get request. Implementing these tasks on a Linux system is also straightforward as it simply requires cron [28] to execute the request. Cron is a standard Linux tool that allows for recurring tasks or jobs to be executed by the system in configurable time intervals. For Cerberus, a cron job is configured to periodically execute a URL request in the Linux shell mimicking the one performed by GAE and reusing the same Java servlet.

3.6 CERBERUS WEB USER INTERFACE

The web user interface component in Cerberus is the one responsible for presenting all the collected data to the user. The entire user interface is developed as a series of Java servlets in the backend that rely heavily on JavaScript for data display on the user's web browser. In addition to using JavaScript, the user interface uses JQuery [9] for many of its presentation elements. JQuery simplifies the task of client side design by abstracting the document manipulation features of JavaScript and also providing convenient interfaces for widget creation and Ajax requests.

Additional features of the user interface include data plotting, user access and a map display. Plotting is achieved using Google's Visualization API [10] which takes a dataset created by the user interface servlets and converts it into a graph on the client side, thus offloading some of the computing effort required by the cloud servers. User access control is currently handled using Google's User Service [11] which provides

simple APIs to authenticate users by using their Google accounts. Finally, environment monitors for which a location has been obtained are all shown on a world map for easy access. The map is generated using Google's Map APIs [12] with markers on each of the device's coordinates. This last feature was not considered in the original design plans and mocks for Cerberus, but it was added as an additional user friendly feature.

Chapter 4: Results

This chapter analyzes the final implementation of Cerberus. It explores the final look and feel of the system as well as its performance in real world situations. It also compares and contrasts its behavior running on different cloud platforms.

4.1 Qualitative Results

The success of an application can be determined by comparing the initial requirements with the end result. In this case, Cerberus can be said to have fulfilled all the specified requirements. The following figures represent screenshots of Cerberus in operation. For the most part, the user interface closely follows the one specified by the mocks. The data shown in the following screenshots was obtained from a live version of Cerberus.

Figure 10: Cerberus Login page

Figure 10 shows the initial page for Cerberus which allows the user to log in to the system. Clicking on the Login button directs the user to a login screen where they can enter their information. For the time being, Google's Users Service [11] is used to handle the authentication. Once successfully authenticated, the user is redirected to the home page.

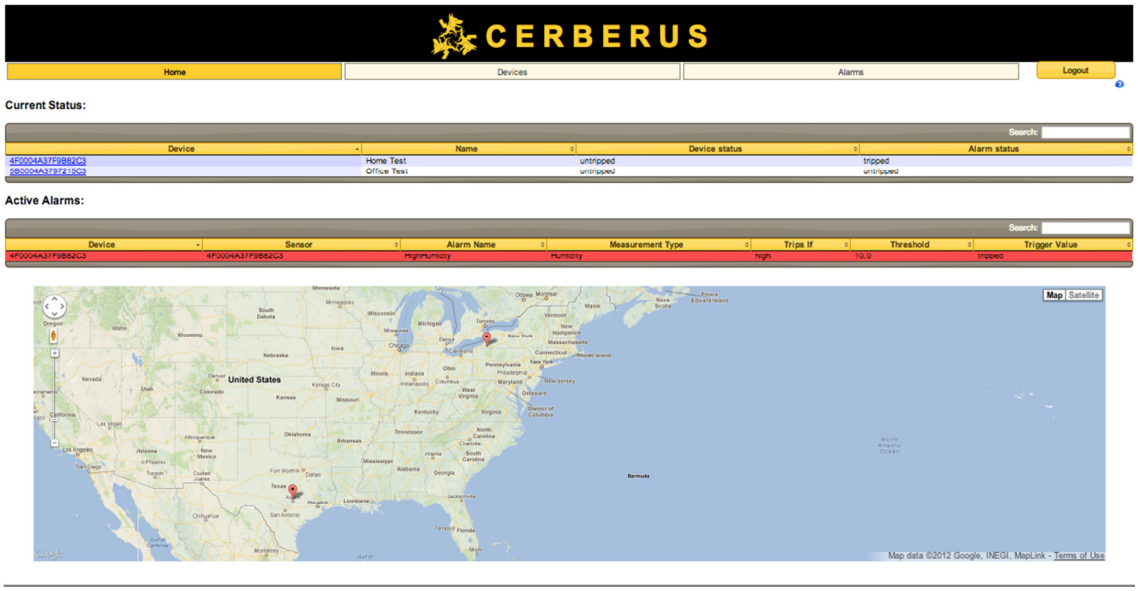


Figure 11: Cerberus home page

Figure 11 shows the Cerberus home page. This page shows all known environment monitors, two in this case, along with any tripped alarms. An extra feature that was not detailed in the original mocks was the addition of the location map. The map is drawn using Google's Map API [12] using any available geolocation data to set the boundary and markers. The markers on the map represent an environment monitor, and can be clicked to navigate to the device page for that monitor. The top navigation menu and all the tables and accordions in the system were made using JQuery UI widgets.

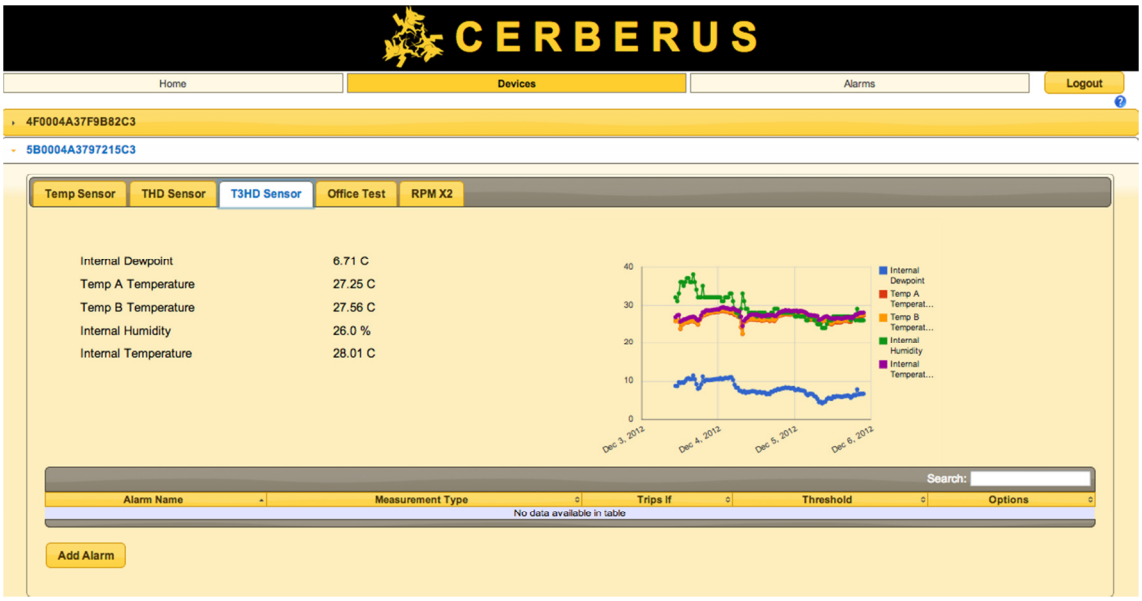


Figure 12: Cerberus sensor view

Figure 12 shows the sensor page for Cerberus. While the layout of this page is slightly different from the mocks, the same functionality remains. The top accordion allows the user to select which environment monitor to view. Each accordion window displays a table with all the sensors associated with that monitor. Clicking on each tab presents the user with the latest measurements for that sensor along with a historical view of the data. While originally there were plans to allow for a user to bring up a dedicated plot consisting of all the data for a single measurement, this feature was deemed unnecessary and it simply overburdened the user interface, so it was removed.

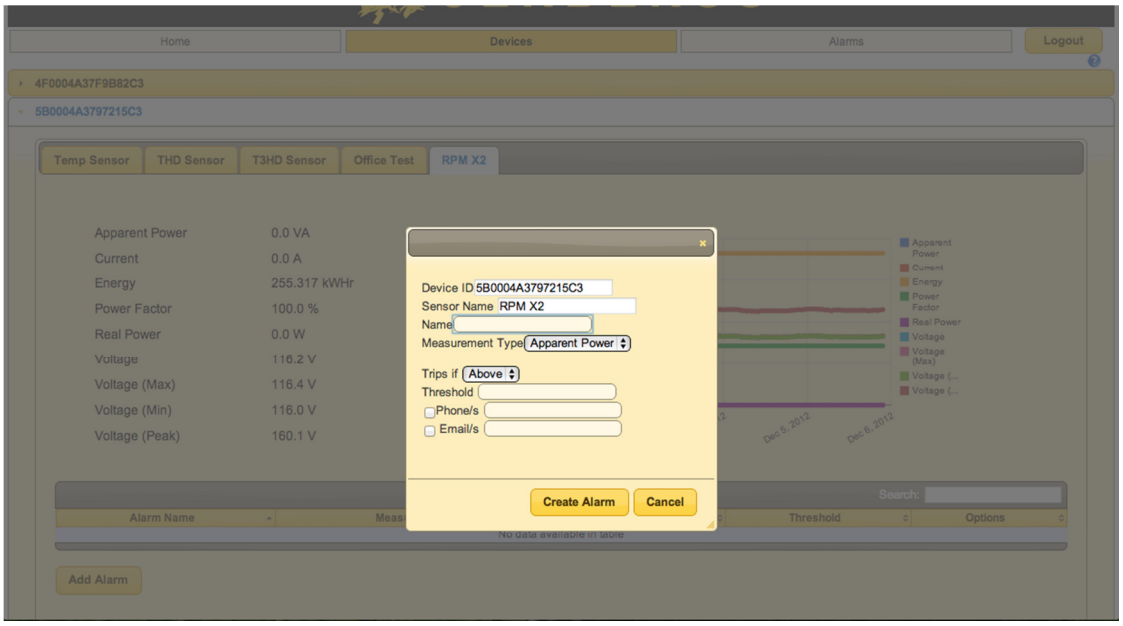
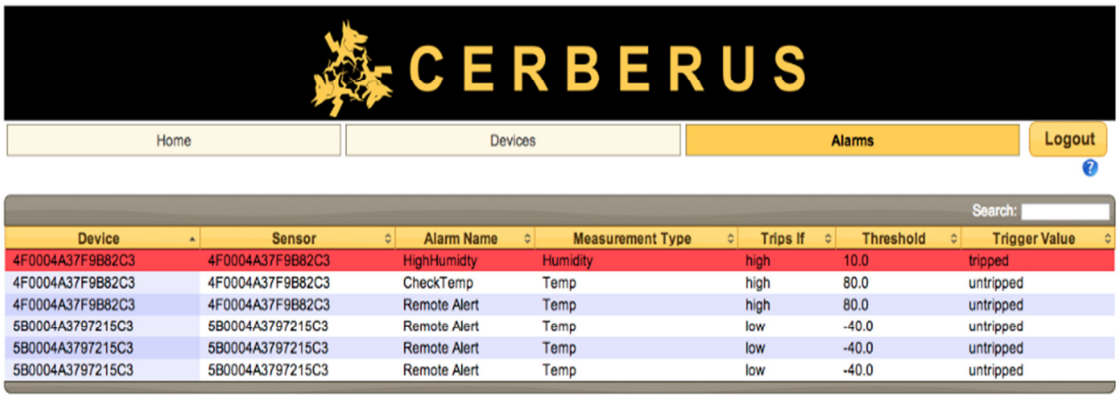


Figure 13: Alarm creation window

Figure 13 shows the alarm creation window. This window is brought up when clicking on the Add Alarm button of the sensor page. As specified, this window allows the user to create an alarm and specify which actions to take.



Device	Sensor	Alarm Name	Measurement Type	Trips If	Threshold	Trigger Value
4F0004A37F9B82C3	4F0004A37F9B82C3	HighHumidty	Humidity	high	10.0	tripped
4F0004A37F9B82C3	4F0004A37F9B82C3	CheckTemp	Temp	high	80.0	untripped
4F0004A37F9B82C3	4F0004A37F9B82C3	Remote Alert	Temp	high	80.0	untripped
5B0004A3797215C3	5B0004A3797215C3	Remote Alert	Temp	low	-40.0	untripped
5B0004A3797215C3	5B0004A3797215C3	Remote Alert	Temp	low	-40.0	untripped
5B0004A3797215C3	5B0004A3797215C3	Remote Alert	Temp	low	-40.0	untripped

Figure 14: Alarm view

Figure 14 shows the alarm page. This page provides a view of all configured alarms on the system as well as their current status. The table can be sorted by any column and has tripped alarms highlighted in red.

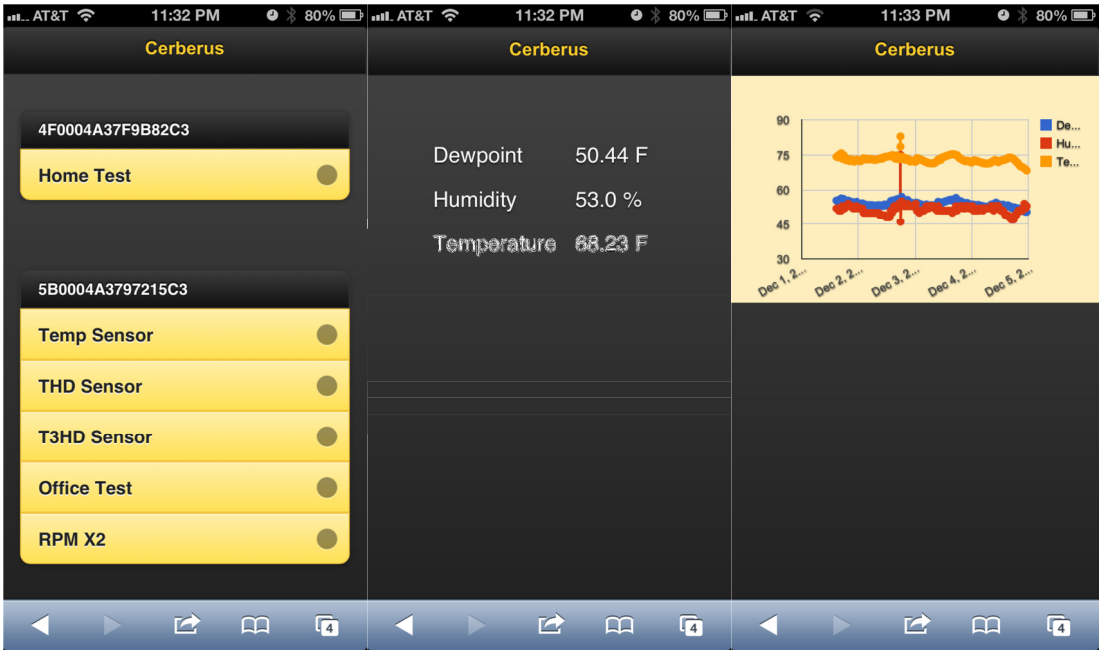


Figure 15: Mobile view

Once the user interface was ready, usability tests revealed that it was not optimal when using a mobile device. So, in addition to the pages shown previously, Cerberus implements a mobile friendly version of the user interface. Figure 15 shows the home screen, a sensor screen and a plot screen for this mobile version. This version of the interface was implemented using JQuery mobile which provides UI widgets optimized for these kinds of applications. As with the original JQuery, this version provides a simple API to easily create the right interface. In order to differentiate mobile users from desktop ones, the Java servlets process the user agent field contained in the HTTP

headers and directs the user to the right page. The location map and alarm view were left out of this interface as they did not quite fit properly.

4.2 Quantitative Results

4.2.1 Impact on Environment Monitors

One of the ideas behind the Cerberus project was to shift some of the work from an environment monitor to a cloud service. While complex data aggregation can be shifted to the cloud, environment monitors must continue to perform all other duties as normal for configurations that do not use Cerberus. Keeping in mind that these products are generally embedded devices, any extra features required to interact with the cloud may cause resource conflicts. As such, these features must minimize their impact on the available resources of the environment monitors. The following sections show the effects of supporting the cloud on various aspects of the environment monitors.

4.2.1.1 Program Memory (ROM)

Additional functionality was required to allow the monitor to establish a connection to the cloud and send the data. Since the XML file was already being generated, no additional code space was required to format the data. In total, the addition of this functionality added 1716 B, representing about 0.33% of the total available code space.

4.2.1.2 Data Memory (RAM)

Additional variables were required to track the state of the cloud communications and when to initiate them. In addition to this, extra memory was allocated to serve as a buffer for the network socket to transmit the data. A total of 2108 B were required, representing about 1.61% of the total available RAM.

4.2.1.3 Processing Time

Table 1 shows time measurements for the data push. The tests were performed with a small data set consisting of just a WatchDog 15 and also with a large data set consisting of the same model but with 4 of the largest expansion sensors attached. The small data set measured 963 B while the large one measured 4060 B. In addition to testing multiple data sets, the tests were run both on the Amazon Web Services (AWS) and on the Google App Engine (GAE) version of Cerberus. A total of 10 tests were performed for each of the four test iterations. The Tx Time column represents the amount of time elapsed between establishing a connection and sending all the data. The Total Time column includes the Tx Time as well as the time required to establish the connection and any additional processing.

	AWS				GAE			
	Small Data Set		Large Data Set		Small Data Set		Large Data Set	
#	Tx Time (ms)	Total Time (ms)	Tx Time (ms)	Total Time (ms)	Tx Time (ms)	Total Time (ms)	Tx Time (ms)	Total Time (ms)
1	59.3	113.59	356.56	408.79	47.22	77.87	204.41	237.06
2	58.93	119.13	373.32	429.71	59.24	59.41	196.21	222.09
3	59.13	114.49	356.92	410.54	34.28	63.55	195.14	221.61
4	60.27	113.46	369.55	428.77	33.61	67.88	198.6	223.76
5	55.74	107.12	363.92	416.93	32.48	63.4	198.1	224.4
6	55.16	109.35	363.82	419.33	28.89	52.87	205.69	233.11
7	60.56	115.07	363.19	421.14	84.71	84.93	194.24	219.32
8	58.81	111.18	424.56	487.99	32.45	60.11	201.48	227.6
9	59.26	112.31	363.95	415.95	29.25	62.13	204.47	233.59
10	57.6	113.91	367.59	420.89	31.71	118.81	194.7	220.9
AVG	58.48	112.96	370.34	426	41.38	71.1	199.3	226.34

Table 1: Data push timing results.

From Table 1 it is clear that the total processing time varies drastically depending on the amount of data sent and the speed of the cloud server. While this is true, the total amount of time did not appear to exceed half a second under the least favorable conditions. Seeing how the network stack for this product allows for multitasking, these results mean that throughput is only affected for a window of half a second every time data is pushed. It is also worth mentioning that the data push function itself does not wait for the acknowledgements from the cloud after the last HTTP packets are sent. This allows this function to exit immediately instead of waiting for the cloud to finish processing the data which would take a greater amount of time.

4.2.1.5 Power

Since the environment monitor is a multitasking system, the processor would be performing some other task, such as measurement readings, when not doing a data push. This would imply that, from the perspective of the processor, no additional power is consumed. The physical network interface, however, does require additional power when sending data. While it is difficult to assess the exact amount of power dedicated to the data push versus normal network activity, an approximation can be made. Using Table 1 as reference, we can assume that the Tx Time is an adequate representation of the amount of time the Ethernet interface must be drawing enough power to transmit. While the Ethernet interface is most likely not powered on for the entire duration, this extra time can account for other factors like the transmission of the TCP SYN packet. Furthermore, the WatchDog 15 uses a LAN8720A [13] Ethernet transceiver which typically uses an extra 23 mA, or 76 mW, when transmitting versus its idle listening state. Using this information, with a worst case average transmit time of 426 ms, we have that each data

push consumes 9.8 mA per second (0.003 mAh) at 3.3 V or 32.38 mW per second (0.009 mWh). For comparison, the idle consumption of the device is 436 mAh at 3.3 V or 1440 mWh.

With relatively low power consumption, the impact of using Cerberus may almost be dismissed in a wired application. However, in an effort to further appeal to the residential or non-technical users, an alternative with wireless communication may be desired. The most straightforward wireless analog for Ethernet communication would be Wi-Fi. In this case, a suitable part could be found in the MRF24WG0MA [14] module. This module consumes and extra 81 mA or 267.3 mW when transmitting data versus its idle listening state. Applying the same calculations as above, we get that a single data push uses 34.51 mA per second (0.0096 mAh) at 3.3 V or 113.87 mW per second (0.032 mWh). While still a relatively low power consumption increase, it is worth mentioning that, in this case, a wireless application requires around 3.52 times more power to do a data push. Even though these devices are not currently optimized for wireless operation, the additional power requirements to use Cerberus would not appear to be a major factor should this conversion be necessary.

4.2.2 Network bandwidth

Several tests were performed with two distinct data sets. One of these consisted of only the data in a WatchDog 15 unit and measured 963 B. The larger data set included not only the WatchDog 15, but also 4 external Remote Power Manager X2 [15] sensors and measured 4060 B. These sensors were selected as they amount to the most data points of any sensor sold by IT Watchdogs. In extreme cases, where many alarms are configured, this block of data could potentially reach 6000 B. Assuming that a single

server can support up to 1000 units and that data is sent every 10 minutes, this would mean that a worst case estimate of the bandwidth necessary to receive all the data would be 36 MB per hour or approximately 26 GB per month.

4.2.3 Comparison of Cloud Platforms

4.2.3.1 Speed

This section explores the differences in timing between the Google App Engine (GAE) and the Amazon Web Services (AWS) versions of Cerberus. For these tests, a micro instance from AWS was used. Each platform was tested both with a small data set of 963 B representing a single WatchDog 15 and a large data set of 4060 B representing a WatchDog 15 with four Remote Power Managers X2 attached sensors. Each test was repeated 10 times and the results monitored with Wireshark [16], a network traffic capture tool. Figure 16 below shows a sample capture from Wireshark.

No.	Time	Source	Destination	Protocol	Info
22	[illegible]	192.168.2.3	173.194.64.141	TCP	csd-mgmt-port > http [SYN] Seq=0 Win=1000 Len=0 MSS=532
24	0.030758000	173.194.64.141	192.168.2.3	TCP	http > csd-mgmt-port [SYN, ACK] Seq=0 Ack=1 Win=62920 Len=0 MSS=1430
25	0.030909000	192.168.2.3	173.194.64.141	TCP	csd-mgmt-port > http [ACK] Seq=1 Ack=1 Win=1000 Len=0
26	0.031531000	192.168.2.3	173.194.64.141	TCP	[TCP segment of a reassembled PDU]
27	0.060750000	173.194.64.141	192.168.2.3	TCP	http > csd-mgmt-port [ACK] Seq=1 Ack=570 Win=63728 Len=0
28	0.063234000	192.168.2.3	173.194.64.141	HTTP	PUT /cerberus HTTP/1.1
35	0.132963000	173.194.64.141	192.168.2.3	TCP	http > csd-mgmt-port [ACK] Seq=1 Ack=965 Win=63728 Len=0
42	0.777724000	173.194.64.141	192.168.2.3	HTTP	HTTP/1.1 200 OK (text/html)
43	0.777729000	173.194.64.141	192.168.2.3	TCP	http > csd-mgmt-port [FIN, ACK] Seq=141 Ack=965 Win=63728 Len=0
44	0.778061000	192.168.2.3	173.194.64.141	TCP	csd-mgmt-port > http [RST, ACK] Seq=965 Ack=1 Win=1000 Len=0

Figure 16: Wireshark traffic capture.

The following tables represent the observed timing for a data push. The description for each column is as follows:

- Setup: Represents the time between the TCP SYN packet sent from the environment monitor and the ACK response from the server. This exchange indicates that a connection to the cloud has been established.

- Done Tx: Represents the time elapsed between the connection being established and the last portion of the data push being sent by the environment monitor.
- Done Rx: Shows the time between the last data packet sent and the ACK of that packet from server. At this point Cerberus has received all the data and the servlet is processing it.
- Done Response: Represents the time elapsed between the final data acknowledgment from the server and the response from the server indicating the operation has been completed.
- Total Time: Shows the entire duration of the exchange from setup to final response.
- Servlet Time: Shows the time for which the servlet was actually active and processing data.

AWS Small Data Set

	Setup (ms)	Done Tx (ms)	Done Rx (ms)	Done Response (ms)	Total Time (ms)	Servlet Time (ms)
1	54.10	59.30	97.20	25.71	236.31	125.00
2	60.15	58.93	94.30	34.06	247.43	127.00
3	55.26	59.13	92.91	18.52	225.81	114.00
4	53.16	60.27	93.68	16.72	223.82	113.00
5	51.16	55.74	89.20	85.42	281.51	176.00
6	54.12	55.16	91.97	42.51	243.76	137.00
7	54.30	60.56	93.38	23.94	232.19	121.00
8	52.51	58.81	98.14	26.67	236.13	125.00
9	52.83	59.26	93.27	26.59	231.96	124.00
10	56.26	57.60	94.48	27.74	236.08	125.00
AVG	54.38	58.48	93.85	32.79	239.50	128.70

Table 2: AWS with small data set.

AWS Large Data Set

	Setup (ms)	Done Tx (ms)	Done Rx (ms)	Done Response (ms)	Total Time (ms)	Servlet Time (ms)
1	52.17	356.56	94.25	397.99	900.98	787.00
2	56.20	373.32	99.83	309.58	838.94	695.00
3	53.50	356.92	93.86	381.55	885.83	706.00
4	59.06	369.55	103.77	215.47	747.85	633.00
5	52.87	363.92	98.40	455.81	971.00	698.00
6	55.31	363.82	94.45	300.92	814.50	627.00
7	57.91	363.19	104.81	393.75	919.66	660.00
8	63.81	424.56	123.54	434.30	1046.21	753.00
9	52.01	363.95	103.63	200.79	720.38	610.00
10	53.16	367.59	97.54	521.87	1040.16	928.00
AVG	55.60	370.34	101.41	361.20	888.55	709.70

Table 3: AWS with large data set.

GAE Small Data Set

	Setup (ms)	Done Tx (ms)	Done Rx (ms)	Done Response (ms)	Total Time (ms)	Servlet Time (ms)
1	30.47	47.22	70.51	668.04	816.24	596.00
2	28.61	59.24	66.60	449.37	575.20	365.00
3	29.07	34.28	70.17	570.07	703.57	480.00
4	34.16	33.61	71.47	316.93	456.18	315.00
5	30.76	32.48	69.73	645.10	778.06	643.00
6	23.96	28.89	65.24	326.24	444.34	326.00
7	45.86	84.71	63.54	319.41	467.66	319.00
8	27.42	32.45	67.36	411.21	538.44	412.00
9	32.67	29.25	65.85	339.22	466.98	336.00
10	86.89	31.71	75.78	363.68	558.06	322.00
AVG	36.99	41.38	68.62	440.93	580.47	411.40

Table 4: GAE with small data set.

GAE Large Data Set

	Setup (ms)	Done Tx (ms)	Done Rx (ms)	Done Response (ms)	Total Time (ms)	Servlet Time (ms)
1	32.51	204.41	73.53	4200.95	4511.40	4194.00
2	26.24	196.21	68.56	3983.55	4374.57	4013.00
3	26.31	195.14	67.96	4343.82	4633.23	4161.00
4	25.58	198.60	68.97	4282.35	4575.51	4101.00
5	26.23	198.10	65.23	4399.90	4689.46	4318.00
6	27.28	205.69	68.32	3453.24	3754.50	3363.00
7	25.17	194.24	67.98	3774.64	4062.03	3731.00
8	26.02	201.48	66.28	3740.03	4033.81	3698.00
9	28.96	204.47	71.46	3634.25	3939.14	3628.00
10	26.14	194.70	66.73	4776.61	5064.18	4768.00
AVG	27.05	199.30	68.50	4058.93	4363.78	3997.50

Table 5: GAE with large data set.

The Tables 2, 3, 4, and 5 show the potential strengths of each platform. On one hand, the GAE alternative had a consistent fast response when establishing a connection. It also had slightly faster transfer rates and the data acknowledgments were handled more promptly. On the other hand, AWS had consistently better times when processing and storing the data. Furthermore, an analysis of the timing indicates that while the GAE alternatives tended to wait for all the data to be present before executing the servlet, the AWS servlets were executed at some point during data reception. While this would seem to indicate that the GAE solution may be slower, it is worth pointing out that its datastore is already separate from the main application and thus requires more time. The version of Cerberus on AWS uses a local database on the instance itself and thus can perform operations faster at the cost of scalability.

The above results are in line with those observed by a program designed to compare cloud solutions, CloudCmp [27]. This program also came to the same conclusions regarding datastore and network speed. As an additional point, GAE's increased network speed advantage also applies to static content, like JavaScript files. This means that, from a user interface perspective, the same results would be observed, that is, AWS has faster datastore operations while GAE has better network performance.

4.2.3.2 Scalability

One of the benefits of using a cloud platform is the ability to maintain performance as demand grows. As such, Cerberus is expected to properly scale to handle varying traffic loads. Each of the tested cloud platforms provides this feature in a different way. AWS, on one hand, provides scalability by allowing an administrator to start multiple application instances on demand. While they also provide services that

initiate new instances based on load, most of the process continues to be manual and must be properly configured by an administrator. GAE, on the other hand, has the advantage of automatically providing scalability. Instead of requiring manual intervention, a GAE application will automatically detect increases in the request load and start extra instances to meet demand. While AWS does not provide the automatic behavior and requires extra configuration, it has the advantage of being much more predictable. Once an instance is characterized, the response of the service can be precisely managed to achieve the desired response times.

Operation	Peak Instance Count	Total Time (s)	Total Requests	Avg Run Time (s)
Data push (1000 units)	14	34.73	1000	6.58
Data Pull (10 users)	2	60.36	1941	0.31
Data Pull (50 users)	7	60.25	4110	0.51

Table 6: GAE scalability tests

Table 6 shows the automatic scaling feature offered by the GAE platform. A total of three tests were performed each starting from an application with no running instances. The results show the number of instances running at the peak of the operation, how long the test ran for, how many total requests were handled and the average time to completion of each request. The first test is a simulation of what would happen if many environmental monitors attempted to push their data at the same time. A computer was used to simulate the data push by spawning 1000 threads in quick succession with each attempting to send data to Cerberus. The result was that GAE ramped up to 14 active instances within the first minute of the operation and all requests succeeded after running an average of 6.58 seconds. While this time is longer than the average measured in the speed tests, the extra delay can be attributed to a combination of the testing computer speed and the increased load on each instance.

The second and third operations depicted in Table 6 attempt to simulate heavy user load. For each one, a computer was set up to spawn a number of threads simulating users. These user threads proceeded to request the home page of Cerberus as many times as possible in one minute. As with the data push test, GAE rapidly ramped up the number of active instances as requests started coming in. This allowed the application to maintain a reasonable service level by providing responses in 0.31 seconds and 0.51 seconds respectively.

An interesting detail observed was that the first set of requests for each data pull operation usually took longer to complete. This delay can probably be attributed to the amount of time needed for an instance to become active and it measured from 3.5 to 4 seconds. Once traffic to the server stabilizes, GAE proceeds to stop any instances that are no longer required. Opposed to the short ramp up period, this shutdown happens after a longer period of time to avoid constant fluctuations in instance numbers as traffic varies. Ultimately, whether the platform provides scalability automatically or by manual configuration, both platforms used by Cerberus are capable of adapting to heavy request volumes and can provide the necessary response characteristics.

4.2.3.3 Availability

The issue of the availability of each cloud platform is a bit difficult to ascertain. Many service outages may be entirely unrelated to the availability and stability of the platform they run on. For instance, some outages may be caused by a fault in the application itself, while other may be related to only a partial outage in a platform. In the case of Cerberus, both the AWS and GAE versions provide similar service level agreements [17] [18] defining an appropriate level of availability for applications. In both

of these cases, a 99.95% uptime is assured with refunds offered for higher levels of unavailability. That being said, AWS has been victim to some high profile disruptions in the past. For instance, the Internet video service Netflix was down during Christmas Eve of 2012 due to an outage in Amazon's cloud platform [19]. In order to mitigate these risks, applications hosted using Amazon's services are advised to be distributed across multiple geographic regions so that a single data center failure is not catastrophic. This is done automatically by GAE.

4.2.3.4 Usability

While the issue of usability is usually one that affects developers more than end users, it must be taken into account to allow for the best maintenance and support. In this respect, the advantage of GAE is the lack of administration duties. This means that the developer does not need to keep track of updates for the various software packages used nor does he need to configure the cloud platform for scalability. On the other hand, the cost of this lack of administration is the lack of flexibility. Not only is the developer locked into Google's platform but he is also restricted to their API. While GAE does provide a rich set of APIs, they are not as rich as what AWS could have access to. A marked example of this is the lack of complex queries in Google's datastore service.

In the case of AWS, the need for administration is offset by the flexibility this provides. There are multiple options for datastore services and they can be as simple or complex as the application requires. Also, the application can be ported to any other cloud platform that offers infrastructure as a service with little to no modifications to the code. Of course, as stated earlier, this does mean that the developer must possess a strong knowledge of server administration to take full advantage of these opportunities.

4.2.3.5 Costs

Table 7 shows a comparison of the costs of each cloud platform. Each item in the table corresponds to different services offered.

- Fronted Instance: Refers to the cloud actively running the application. In the case of the GAE platform, this means that requests have been made and are being handled and extra instances are added based on demand through their automatic scaling. For AWS, this number refers to a running instance which can have varying degrees of resources, hence the variable price.
- Storage: Mechanisms to store application related data. GAE provides 5GB for free, which is enough to store the entire application code, while AWS requires an EBS image to keep not only the application, but the operating system and configuration as well.
- Bandwidth in and out: This refers to transferring data from the Internet to the cloud server and vice versa. The data push from environmental monitors would count as bandwidth in and is free of charge. Seeing how the user interface would count as bandwidth out, precautions should be taken to minimize the amount of data transferred. This objective would also be in line with making the user interface responsive by not transferring large sets of data unnecessarily.
- Datastore Instance: In the case of AWS, if we desire scalable and consistent data, we would require a dedicated instance for the datastore. These instances are specially tailored for this purpose but are charged separately. GAE does not require this as their datastore service is always available.

- Datastore Storage: This represents the amount charged based on the volume of data in the datastore. In the case of Cerberus, the bulk of the stored data is in the historic values, so they should be optimized.
- Datastore Reads and Writes: This is the cost for performing reads and writes on the datastore. Since AWS uses either a local database or a dedicated instance, and transfer of data within AWS is free, these operations have no cost. For GAE, the cost of reads may become large as more historic data is read from the datastore.
- Load Balancing: Represents the cost for balancing incoming requests among active instances. GAE does this automatically and for free while AWS charges an amount for running the balancer itself, plus a cost for each GB that is processed by the balancer.
- Automatic Scaling: The cost to achieve automatic scaling of active instances. Once again this service is automatic and free from GAE while AWS requires use of their CloudWatch service which carries a cost per instance.
- Email: The cost to send out alert emails. Sending emails through GAE carries a cost per recipient. Being a Linux server, the AWS application can use any email program available and the costs would be factored into the outgoing bandwidth.

From Table 7, it can be seen that the cost can vary drastically depending on usage. A final decision would have to depend on the adoption rate and requests, which would drive the size and number of active instances required. Also, while the datastore storage costs are fairly similar, the cost per operation of GAE is a definite disadvantage. Under the current scheme, which could be drastically improved, plotting historic data for half a year for a single sensor would require tens of thousands of read operations which would quickly add up in cost. These numbers also show that the approximate required

bandwidth of 26 GB per month would have no impact other than the instances required to process that data as incoming bandwidth is free.

	GAE	AWS
Fronted Instance	$0.08/hr	$0.02/hr-$0.48/hr
Storage	$0.13/GB/month	$0.10/GB/month
Bandwidth (IN)	free	free
Bandwidth (OUT)	$0.12/GB	$0.12/GB
Datastore (Instance)	free	$0.025/hr-$2.215/hr
Datastore (Storage)	$0.18/GB/month	$0.125/GB/month
Datastore (Writes)	$0.09/100k	free
Datastore (Reads)	$0.06/100k	free
Load Balancing	free	$0.025/hr + $0.008/GB
Automatic Scaling	free	$3.5/instance/month
Email	$0.01/recipient	free

Table 7: Cloud platform costs

4.2.4 Engineering effort

The effort involved in maintaining a large application usually represents a significant portion of its costs. As such, the following numbers provide a quantification of this complexity.

4.2.4.1 Lines of Code

The source base for Cerberus contains:

- 22 java files with a total 2640 lines of code on GAE and 2496 lines of code on AWS.
- 4 JavaScript files with a total of 213 lines.
- 5 CSS files with a total 129 lines.
- All files, including libraries, add up to 33127 lines on GAE and 32983 on AWS.

4.2.4.2 Source Control

The two developed versions of Cerberus are kept in separate Subversion [20] repositories. Most of the development was performed on the Google App Engine version of Cerberus as it was the first one created. In total, this version accrued 44 code commits. The second version of Cerberus, the Linux variant on a Tomcat web server, ended up with only a fraction of the commits, totaling 5. It may be worth mentioning that best practices were not always followed as there were several long periods of time with numerous changes that went without a code commit.

Chapter 5: Conclusions

5.1 SUMMARY

This report presented Cerberus, a cloud-based environmental data aggregator. It has explored the vision and ideas behind its conception as well as its requirements and design. The application was successfully completed on two different cloud platforms, Google App Engine and a Linux server with Tomcat webserver running on Amazon Web Services. The strengths and weaknesses of each platform were explored as well as their interactions with the environment monitors they interface to. The impact of supporting such a service on these environment monitors was also explored.

In light of the results it can be concluded that supporting Cerberus consumes a relatively small amount of resources on environment monitors and thus was feasible to implement. This can be considered to validate the original objectives of reducing the workload on an embedded system by moving the aggregation duties to the cloud.

The final product was able to successfully perform all the required tasks. Cerberus was capable of receiving, processing and storing data from environment monitors as well as sending out alerts when necessary. Furthermore, it did so while providing adequate levels of scalability, availability, maintainability, and response time while incurring reasonable costs. More importantly, it also proved to be a convenient aggregation portal for environmental data that could be used by any kind of user regardless of their physical location.

While Cerberus will require additional polishing before it can be marketed to end customers, it has proven that the design will accomplish the required tasks. The next objectives are to further improve, and potentially expand, the capabilities of Cerberus in an effort to bring its benefits to an ever expanding market. The tools and concepts behind Cerberus and cloud computing can be used to help many markets that are falling behind

the times. As seen here, they have been shown to provide inexpensive modernization paths and alternatives that far exceed the original designs.

5.2 LESSONS LEARNED

- The user interface development was observed to take up all available time. There are points in time where the interface is simply good enough and obsessing over it produces rapidly diminishing returns.
- The data push feature of the environment monitors remained untested down to the end and it turned out not to properly work with the Cerberus web servers. As the product would not have worked without this feature, having to fix this functionality at the last minute was not prudent. In retrospect, it seems obvious that major features be fully tested before moving on to lower priorities, but sometimes people must be reminded of this.
- There is a lot of high quality software out there that can be used to speed up development. Cerberus attempts to leverage as many of these tools as possible and the project has benefited greatly from it.
- Proper source control usage should not be neglected. Fortunately there were no incidents of lost code during the development of Cerberus, but this good fortune could have run out at any moment.
- The Amazon Web Services platform does not provide any way to control and limit costs incurred by a running application. As such, special care should be taken to ensure that the right security settings are employed so that the service is not exploited. Failure to do so resulted in an issue where an unknown source caused the application to serve out Terabytes worth of data in a couple days. This

resulted in charges in excess of $600. Fortunately, Amazon's security department caught the suspicious activity and notified the administrators so that the situation could be remedied.

5.3 FUTURE WORK

5.3.1 Feature Enrichment

- Environment monitor should be configurable from the cloud. While data being made available from the monitors is a very useful feature, it would be better if a user could log into Cerberus, configure the device and then have all the changes be pushed back to the monitors. In order to accomplish this, the devices would have to query Cerberus every time they push data, or perhaps even more often than that, and then download any changes. This feature could be expanded to include pushing down firmware updates and thus make access to the embedded web server by the user completely unnecessary.
- As Cerberus collects data for extended periods of time, the historic plots become slower and slower to generate. A solution to this potential issue would be to decimate the data before it is sent client side. Additionally, plots could be enhanced to allow for custom time ranges. This way, the amount of data Cerberus would have to serve for the initial plot view would be minimal but, on request, the user could zoom in to some time period and obtain more detailed data for that interval.
- Additional actions could be added to perform in case of alarm. These actions could include phone calls, posting of data to some other website or even sending commands, such as shutdown, to a remote computer.

- There are many other features provided by environment monitors and they could be made available through Cerberus. In the case of the particular brand of monitors used, some sensors can monitor and control power outlets. Having the ability to switch them remotely could be valuable but it would require UI changes on Cerberus as well as the ability to push data back to the monitors.
- The mobile interface is relatively lacking in data. It could be improved to show the location map or table of alarms.

5.3.2 Alternate Data Sources

The current implementation of Cerberus is designed to work only with IT Watchdogs products. Furthermore, it only applies to environment monitors. An area of interest would be to expand its capabilities to accept other brands of monitors and perhaps even broaden the type of devices that can interface to it. In order to accomplish this goal, Cerberus could be modified to parse the data formats for other devices and to display the data correctly. Unfortunately, this expansion would also require the cooperation of other companies as their devices may not have the capability to push data to a remote server. So, while the objective to expand Cerberus to alternate data sources is not necessarily difficult, it would require the participation of multiple entities which may have their own objectives.

5.3.3 Marketability

As a first step to commercialize Cerberus, additional polishing and quality assurance needs to take place. Further testing of the scalability of the system must take place to ensure that the user experience will be acceptable. Afterwards, the current

implementation of Cerberus would be best marketed alongside products from IT Watchdogs and with their cooperation. Given the relatively low recurring costs of the platform, an initial idea would be to provide a short trial period for Cerberus with the sale of each environment monitor. Once this period expires the user may opt into a monthly subscription with a relatively low cost. This cost would be proportional to the number of devices monitored by the user. This business model would be nothing new to the non-technical audience Cerberus is best suited for and is meant to replace the cost of figuring out the proper configurations.

5.4 Related Work

There are multiple ways in which data can be aggregated from network connected devices. A classic alternative is to use one of the many SNMP data aggregators available. Software such as HP OpenView [21], SolarWinds products [22], Spiceworks [23], or many others can poll devices for their current measurements and present them in an intuitive and central manner. The drawback of these products is that the customer must be knowledgeable in technical matters and that the data is only obtainable on a local network unless the user goes through complex configuration steps.

Directly related to the field of environment monitors, many manufacturers of these products provide their own aggregation software. As an example, Avtech [24] has their own software solutions for this purpose [25]. As with the SNMP aggregation software, this solution requires a server sharing the same network as the monitored devices. A more cloud-based approach can be found in FlashRF's Data Central [26]. Tailored to their wireless temperature and humidity sensors, this solution provides the same kind of advantages as Cerberus except that it requires a PC running their software

to do the data push instead of allowing the sensors themselves to do so. This system provides alarm notifications via Email or SMS as well as historic data and plots. An interesting feature provided by Data Central is the ability to create and configure floor plans where sensors can be viewed in a user friendly manner.

References

[1] IT Watchdogs, Watchdog 15 Product Page. 2013. Available: http://www.itwatchdogs.com/product-detail-watchdog_15-71.html

[2] IT Watchdogs. 2013. Available: http://www.itwatchdogs.com/

[3] Google. Google App Engine. 2013. Available: https://developers.google.com/appengine/

[4] The Apache Software Foundation. Apache Tomcat. 2013. Available: http://tomcat.apache.org/

[5] Amazon. Amazon Web Services. 2013. Available: http://aws.amazon.com/

[6] Geobytes, Inc. IP Address Locator. 2003. Available: http://www.geobytes.com/iplocator.htm

[7] Twilio, Inc. Twilio SMS. 2013. Available: http://www.twilio.com/sms

[8] Google. Google App Engine Java Datastore API. 2013. Available: https://developers.google.com/appengine/docs/java/datastore/

[9] The JQuery Foundation. 2013. Available: http://jquery.com/

[10] Google. Google Visualization API Reference. 2013. Available: https://developers.google.com/chart/interactive/docs/reference

[11] Google. Using the User Service. 2013. Available: https://developers.google.com/appengine/docs/java/gettingstarted/usingusers

[12] Google. Maps. 2013. Available: https://developers.google.com/maps/

[13] SMSC. LAN8710A, LAN8720A. 2013. Available: http://www.smsc.com/Products/Ethernet_and_Embedded_Networking/Ethernet_Transceivers/LAN8710A_LAN8720A

[14] Microchip Technology Inc. MRF24WG0MAIn. 2013. Available: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en559196

[15] IT Watchdogs. Remote Power Manager X2 Product Page. 2013. Available: http://www.itwatchdogs.com/product-detail-remote_power_manager_x2-61.html

[16] Wireshark Foundation. 2013. Available: http://www.wireshark.org/

[17] Amazon Web Services. Amazon EC2 Service Level Agreement. 2013. Available: http://aws.amazon.com/ec2-sla/

[18] Google. App Engine Service Level Agreement. 2013. Available: https://developers.google.com/appengine/sla

[19] J. Roettgers. Netflix is down: AWS outage takes down service on some devices. Gigaom 2012. Available: http://gigaom.com/2012/12/24/netflix-down-xmas-eve/

[20] The Apache Software Foundation. Apache Subversion. 2013. Available: http://subversion.apache.org/

[21] Hewlett-Packard Development Company, L.P. Enterprise Software. 2013. Available: http://www8.hp.com/us/en/software/enterprise-software.html

[22] SolarWinds. 2013. Available: http://www.solarwinds.com/

[23] Spiceworks, Inc. 2013. Available: http://www.spiceworks.com/

[24] AVTECH Software, Inc. 2013. Available: http://avtech.com

[25] AVTECH Software, Inc. Device ManageR. 2013. Available: http://avtech.com/Products/Software/

[26] DeltaTRAK. FlashRF Data Central. 2011. Available: https://flashrf.com/default.aspx

[27] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Comparing Public Cloud Providers. 10th ACM SIGCOMM conference on Internet measurement. 2010. Available: http://conferences.sigcomm.org/imc/2010/papers/p1.pdf

[28] P. Vixie. cron - daemon to execute scheduled commands (ISC Cron V4.1). 1996. Available: http://www.linuxmanpages.com/man8/cron.8.php

[29] J. Brittain. Tomcat: The Definitive Guide, Second Edition. O'Reilly Media, Inc. 2008.

[30] D. Flanagan. JQuery Pocket Reference. O'Reilly Media, Inc. 2011.

[31] D. Gourley, B. Totty. HTTP: The Definitive Guide. O'Reilly Media, Inc. 2002.